# Unobtrusive JavaScript

# The six global DOM objects



| name | description |
|---|---|
| document | current HTML page and its content |
| history | list of pages the user has visited |
| location | URL of the current HTML page |
| navigator | info about the web browser you are using |
| screen | info about the screen area occupied by the browser |
| window | the browser window |

# The window object



- *the entire browser window; the top-level object in DOM hierarchy*
- technically, all global code and variables become part of the window object properties:
  - document, history, location, name
- methods:
  - alert, confirm, prompt (popup boxes)
  - `setInterval, setTimeout clearInterval, clearTimeout` (timers)
  - `open, close` (popping up new browser windows)
  - `blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo`

# The document object



- *the current web page and the elements inside it*
- properties:
  - `anchors, body, cookie, domain, forms, images, links, referrer, title, URL`
- methods:
  - `getElementById`
  - `getElementsByName`
  - `getElementsByTagName`
  - `close, open, write, writeln`

# The location object



- *the URL of the current web page*
- properties:
  - `host, hostname, href, pathname, port, protocol, search`
- methods:
  - `assign, reload, replace`

# The navigator object



- *information about the web browser application*
- properties:
  - `appName, appVersion, browserLanguage, cookieEnabled, platform, userAgent`
- Some web programmers examine the navigator object to see what browser is being used, and write browser-specific scripts and

```
if (navigator.appName === "Microsoft Internet Explorer") {
...
```

JS

# The screen object



- *information about the client's display screen*
- properties:
  - `availHeight, availWidth, colorDepth, height, pixelDepth, width`

# The history object



- the list of sites the browser has visited in this window
- properties:
  - `length`
- methods:
  - `back, forward, go`
- sometimes the browser won't let scripts view `history` properties, for security

# Unobtrusive JavaScript



- JavaScript event code seen previously was *obtrusive*, in the HTML; this is bad style
- now we'll see how to write unobtrusive JavaScript code
  - HTML with minimal JavaScript inside
  - uses the DOM to attach and execute all JavaScript functions

# Unobtrusive JavaScript



- allows separation of web site into 3 major categories:
  - content (HTML) - what is it?
  - presentation (CSS) - how does it look?
  - behavior (JavaScript) - how does it respond to user interaction?

# Obtrusive event handlers (bad)



```
<button id="ok" onclick="okayClick();">OK</button>
```
HTML

```
// called when OK button is clicked
function okayClick() {
	alert("booyah");
}
```
JS

- this is bad style (HTML is cluttered with JS code)
- goal: remove all JavaScript code from the HTML body

# Attaching an event handler in JavaScript code



```
// where element is a DOM element object
element.event = function;
```
JS

```
$("ok").onclick = okayClick;
```
JS

- it is legal to attach event handlers to elements' DOM objects in your JavaScript code
  - notice that you do not put parentheses after the function's name
- this is better style than attaching them in the HTML
- Where should we put the above code?

# When does my code run?



```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```
HTML

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```
JS

- your file's JS code runs the moment the browser loads the script tag
  - any variables are declared immediately
  - any functions are declared but not called, unless your global code explicitly calls them

# When does my code run?



```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body> ... </body>
```
HTML

```
// global code
var x = 3;
function f(n) { return n + 1; }
function g(n) { return n - 1; }
x = f(x);
```
JS

- at this point in time, the browser has not yet read your page's body
  - none of the DOM objects for tags on the page have been created

# A failed attempt at being unobtrusive



```
<head>
<script src="myfile.js" type="text/javascript"></script>
</head>
<body>
<div><button id="ok">OK</button></div>
```
HTML

```
// global code
$("ok").onclick = okayClick; // error: $("ok") is null
```
JS

- problem: global JS code runs the moment the script is loaded
- script in head is processed before page's body has loaded
  - no elements are available yet or can be accessed yet via the DOM

# The `window.onload` event



```
// this will run once the page has finished loading
function functionName() {
	element.event = functionName;
	element.event = functionName;
...
}
window.onload = functionName; // global code
```

JS

- we want to attach our event handlers right after the page is done loading
  - there is a global event called `window.onload` event that occurs at that moment
- in `window.onload` handler we attach all the other handlers to run when events occur

# An unobtrusive event handler



```
<!-- look Ma, no JavaScript! -->
<button id="ok">OK</button>
```

HTML

```
// called when page loads; sets up event handlers
function pageLoad() {
	$("ok").onclick = okayClick;
}
function okayClick() {
	alert("booyah");
}
window.onload = pageLoad; // global code
```

JS

# Common unobtrusive JS errors



```
window.onload = pageLoad();
window.onload = pageLoad;
okButton.onclick = okayClick();
okButton.onclick = okayClick;
```
JS

- event names are all lowercase, not capitalized like most variables

```
window.onLoad = pageLoad;
window.onload = pageLoad;
```
JS

# Anonymous functions



```
function(parameters) {
        statements;
}
```
JS

- JavaScript allows you to declare anonymous functions
- quickly creates a function without giving it a name
- can be stored as a variable, attached as an event handler, etc.

# Anonymous function example



```
window.onload = function() {
	var okButton = document.getElementById("ok");
	okButton.onclick = okayClick;
};
function okayClick() {
	alert("booyah");
}
```

JS

# The keyword `this`



```
this.fieldName // access field
this.fieldName = value; // modify field
this.methodName(parameters); // call method
```

JS

- all JavaScript code actually runs inside of an object
- by default, code runs inside the global window object
  - all global variables and functions you declare become part of window
- the this keyword refers to the current object

# The keyword `this`



```
function pageLoad() {
	$("ok").onclick = okayClick; // bound to okButton
here
}
function okayClick() { // okayClick knows what DOM object
	this.innerHTML = "booyah"; // it was called on
}
window.onload = pageLoad;
```

JS

- event handlers attached unobtrusively are **bound** to the element
- inside the handler, that element becomes this (rather than the window)

# Fixing redundant code with `this`



```
<fieldset>
	<label><input type="radio" name="ducks"
value="Huey" /> Huey</label>
	<label><input type="radio" name="ducks"
value="Dewey" /> Dewey</label>
	<label><input type="radio" name="ducks"
value="Louie" /> Louie</label>
</fieldset>
```

HTML

```
function processDucks() {
	if ($("huey").checked) {
		alert("Huey is checked!");
	} else if ($("dewey").checked) {
		alert("Dewey is checked!");
	} else {
		alert("Louie is checked!");
	}
	alert(this.value + " is checked!");
}
```

JS

# Example: Tip Calculator



```
<h1>Tip Calculator</h1>
<div>
        $<input id="subtotal" type="text" size= "5" /> subtotal
<br />
        <button id="tenpercent">10%</button>
        <button id="fifteenpercent"> 15%</button>
        <button id="eighteenpercent"> 18%</button>

        <span id="total"></span>
</div>
```

*HTML*

```
window.onload = function() {
        $("tenpercent").onclick = computeTip;
        }
function computeTip{
        var subtotal = parseFloat($("subtotal").value);
        var tipAmount = subtotal*0.1;//Add this code
        $("total").innerHTML = "Tip: $" + tipAmount;
}
```

*JS*